Architecture and Design Document

Hangout

Revision History

Date	Description	Author	Comments
10/14/2019	Initial Draft	Antonio Sanchez, Brandon Pitcher, Eric Curlett, Dean Vo, Ashley Williams	Initial draft of document
10/15/2019	v1.5	Antonio Sanchez	Added Scope, description, and trade-off analysis

Document Approval

Feature Name	Printed Name	Title	Date

Table of Contents

Introduction

1.1 Purpose

This architecture and design document is intended display the overview of implementation of *Hangout* at every level.

1.2 Scope

This design document is meant to provide an overview of the structure of our webapp. This document also includes the tradeoff analysis between different frameworks to choose from. UML Diagrams and Sequence Diagrams are included to show how each component interacts with each other.

1.3 Intended Audience

Hangout is intended as a lightweight web app that users will be able to quickly and efficiently meet with others of similar interests. Hangout is designed to be used by all types of users who might use social media, and thus must be designed to be accessible and quick over detailed and complicated.

1.4 Overview

Hangout will use a Decision Tree learning model inorder to serve relevant ads as well as suggestions for Hangout™s that we believe the users will likely respond well to and be interested in.

The server side code will be written in NodeJS, which is sufficiently lightweight and flexible for our needs as well as being relatively simple

We will have an Apache Web Server running a virtual console of Debian

React will be used as the front end. React is fast, scalable, and highly testable. It also is reportedly also very simple and easy to learn, which is appealing as we have limited time and capacity.

1.5 Tradeoff analysis

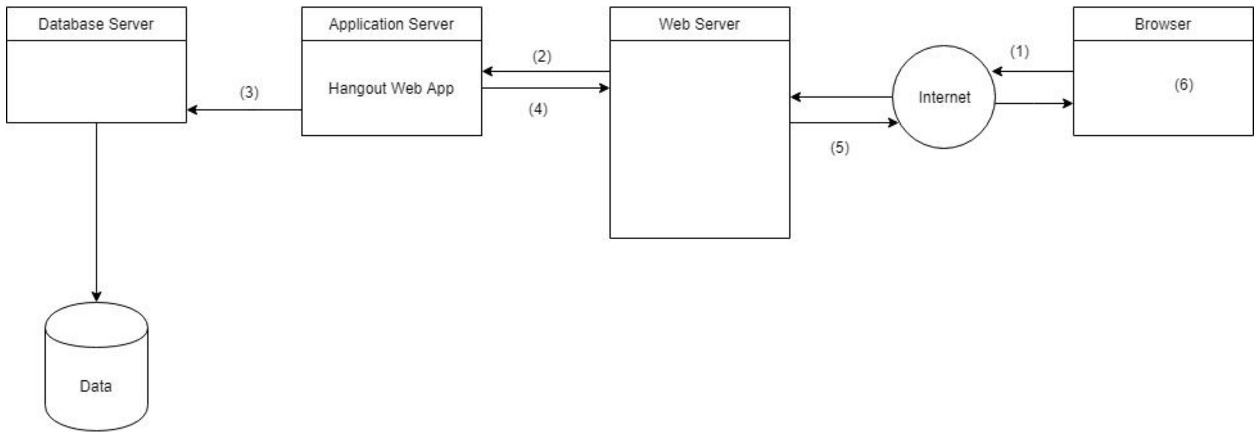
Front end	Pros	Cons
React	Familiarity, easy to learn, Fast, supports server-side rendering, Functional programming	Less standardized, Less class based, Mixes templating with JSX logic
Angular	Supports Typescript, prebuilt npm libraries, One way data binding	Variety of structures, slower than others

Vue	Adaptable, offers strong integration tools, allows for large templates, low weight	Small market share means finding resources is harder. Less global experience as well as team experience.

Backend	Pros	Cons
Node	Non-blocking I/O, single language, Flexible, familiarity, JSON focused, speed	Single threaded, Complexity
PHP	Massively popular and well supported, supports relational databases	Slower, combines html with it resulting in confusion, not component modular
Java	Super familiar, powerful, cross platform	Most server plugins require payment
Linode (virtual server)	affordable, transparent, scalable, reliable, full access to server, great customer service w/ phone line support, predictable pricing, open cloud (no vendor lock in), IP address of virtual server does not change	only supports Linux operating systems\
Express	most popular web framework on NPM, provides simple user authentication, allows for login using social media, opensource	single threaded framework, poor scalablity

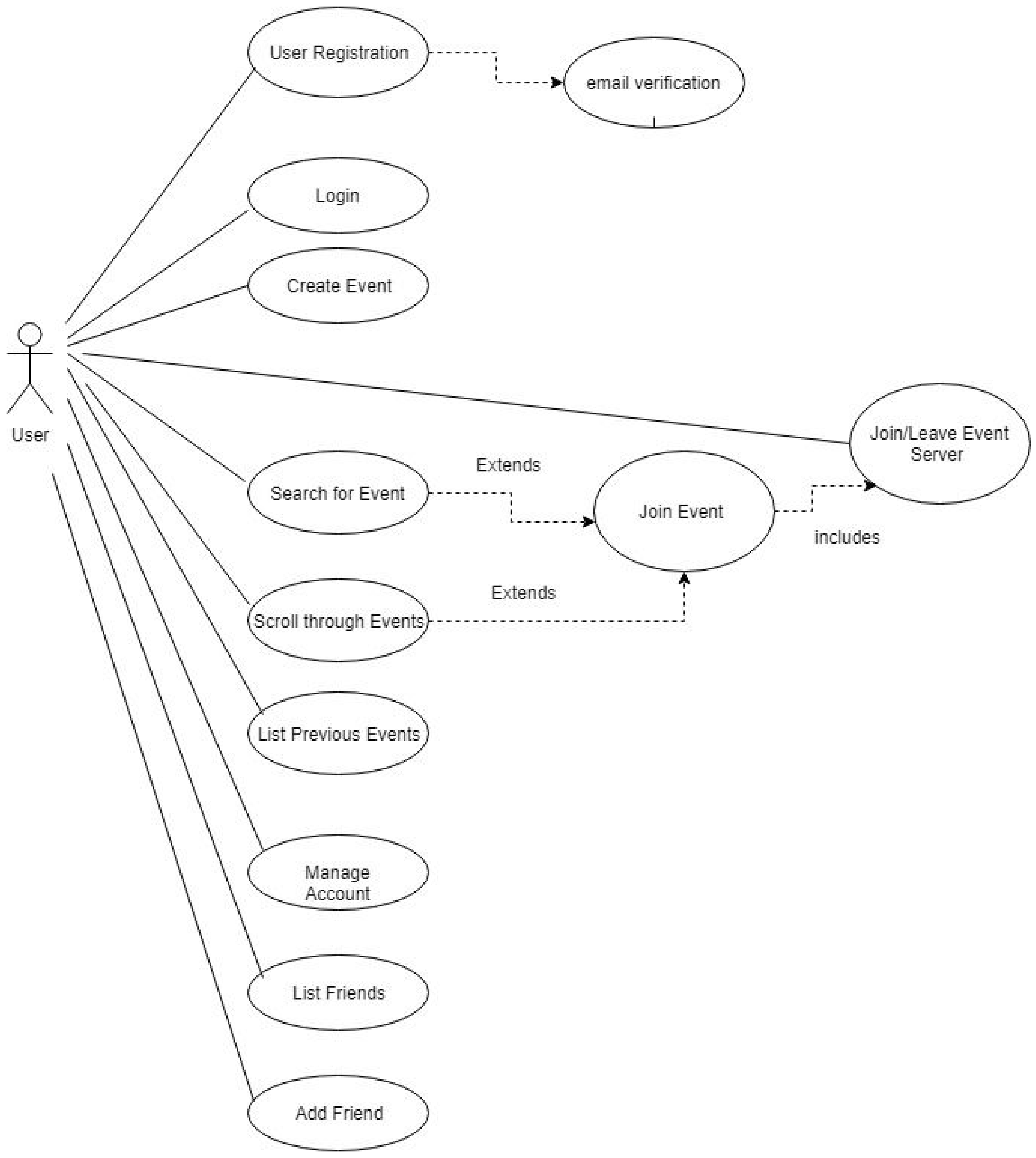
Architecture Details

2.1 Top Level Architecture System Diagram



The diagram above is a block view of the *Hangout* system, displaying the different interactions between the modules. The numbered arrows show the flow of control between the modules. It behaves like a standard web app with the flow of control going from browser (1) to web server to our web app, and finally to our database server that stores our user info .

2.2 Use Cases


User
User Registration
email verification
Login
Create Event
Join/Leave Event Server
Search for Event
Extends
Join Event
includes
Extends
Scroll through Events
List Previous Events
Manage Account
List Friends
Add Friend

2.2.1 Use Case UC1

2.2.1.1 *Objective*

Registration

2.2.1.2 *Priority*

High

2.2.1.3 *Actors*

End-User

2.2.1.4 *Pre-conditions*

The user is connected to the internet and navigates to the *Hangout* website on their browser.

2.2.1.5 *Post-conditions*

The user is now a member of *Hangout.*

2.2.2 Use Case UC2

2.2.2.1 *Objective*

Verification

2.2.2.2 *Priority*

Medium

2.2.2.3 *Actors*

End-User

2.2.2.4 *Pre-conditions*

The user is registering to *Hangout* as a new user.

2.2.2.5 *Post-conditions*

User is now a registered user of *Hangout* and create/search for Events and update their account.

2.2.3 Use Case UC3

2.2.3.1 *Objective*

Login

2.2.3.2 *Priority*

High

2.2.3.3 *Actors*

End-User

2.2.3.4 *Pre-conditions*

The user is connected to the internet and navigates to the *Hangout* website on their browser.

2.2.3.5 *Post-conditions*

User signs in and has access to the *Hangout* home page.

2.2.4 Use Case UC4

2.2.4.1 *Objective*

Create Event

2.2.4.2 *Priority*

High

2.2.4.3 *Actors*

End-User

2.2.4.4 *Pre-conditions*

A user is logged into *Hangout*.

2.2.4.5 *Post-conditions*

User has posted a new Event to the site.

2.2.5 Use Case UC5

2.2.5.1 *Objective*

Search Event

2.2.5.2 *Priority*

High

2.2.5.3 *Actors*

End-User

2.2.5.4 *Pre-conditions*

User logged into *Hangout*

2.2.5.5 *Post-conditions*

User is given a list of Events that match their search criteria.

2.2.6 Use Case UC6

2.2.6.1 *Objective*

Join Event

2.2.6.2 *Priority*

High

2.2.6.3 *Actors*

End-User

2.2.6.4 *Pre-conditions*

User is given an event list that matches his search criteria.

2.2.6.5 *Post-conditions*

User Joins an event.

2.2.7 Use Case UC7

2.2.7.1 *Objective*

Scroll through events

2.2.7.2 *Priority*

Medium

2.2.7.3 *Actors*

End-User

2.2.7.4 *Pre-conditions*

User can navigate through events held on a given day.

2.2.7.5 *Post-conditions*

The user is now a member of *Hangout.*

2.2.8 Use Case UC8

2.2.8.1 *Objective*

Join/Leave Event Server

2.2.8.2 *Priority*

Medium

2.2.8.3 *Actors*

End-User

2.2.8.4 *Pre-conditions*

User joins event.

2.2.8.5 *Post-conditions*

User is added to a chat with other users that joined the same event. User can also leave or mute the chat.

2.2.9 Use Case UC9

2.2.9.1 *Objective*

List Previous Events

2.2.9.2 *Priority*

Low

2.2.9.3 *Actors*

End-User

2.2.9.4 *Pre-conditions*

User is at their home page.

2.2.9.5 *Post-conditions*

User can see previous events they participated in.

2.2.10 Use Case UC10

2.2.10.1 *Objective*

Manage Account

2.2.10.2 *Priority*

High

2.2.10.3 *Actors*

End-User

2.2.10.4 *Pre-conditions*

User is logged in and in their home page.

2.2.10.5 *Post-conditions*

User account has been updated as per their requirements.

2.2.11 Use Case UC11

2.2.11.1 *Objective*

List Friends

2.2.11.2 *Priority*

Low

2.2.11.3 *Actors*

End-User

2.2.11.4 *Pre-conditions*

2.2.11.5 *Post-conditions*

2.2.12 Use Case UC12

2.2.12.1 *Objective*

Add Friends

2.2.12.2 *Priority*

Low

2.2.12.3 *Actors*

End-User

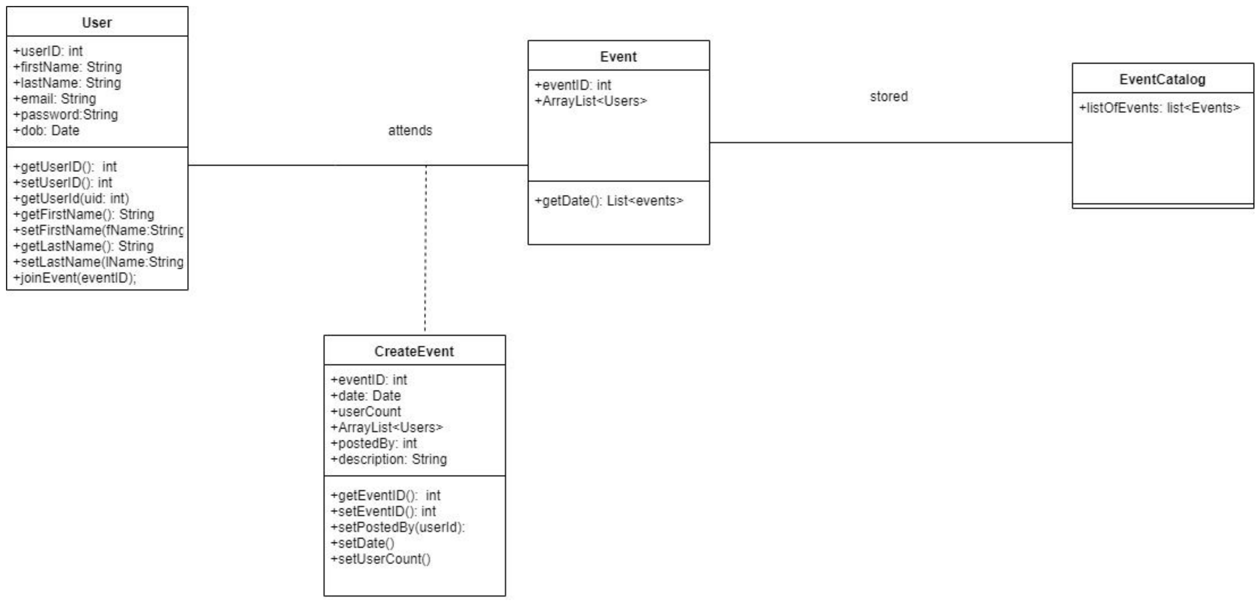
2.2.12.4 *Pre-conditions*

User has finished an event.

2.2.12.5 *Post-conditions*

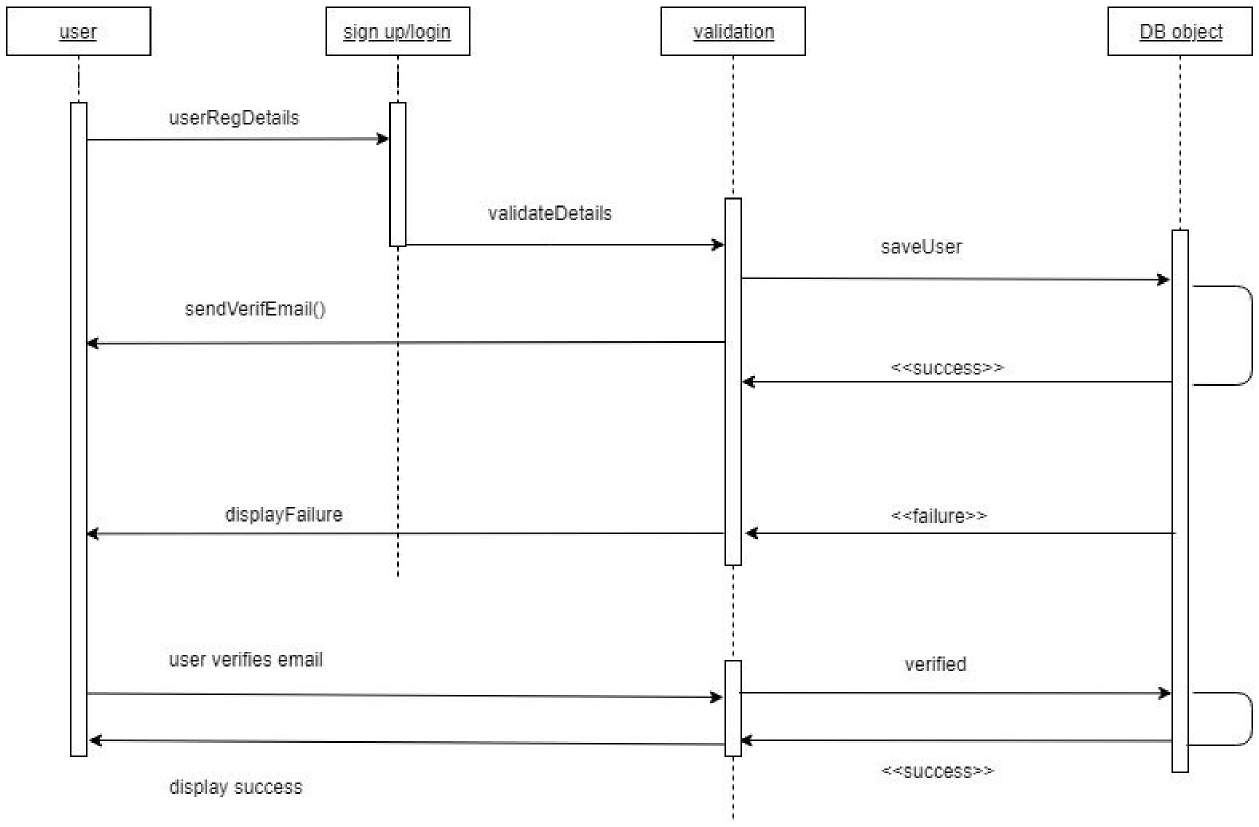
User can add other Users from past events to friends list.

2.3 Class Diagram

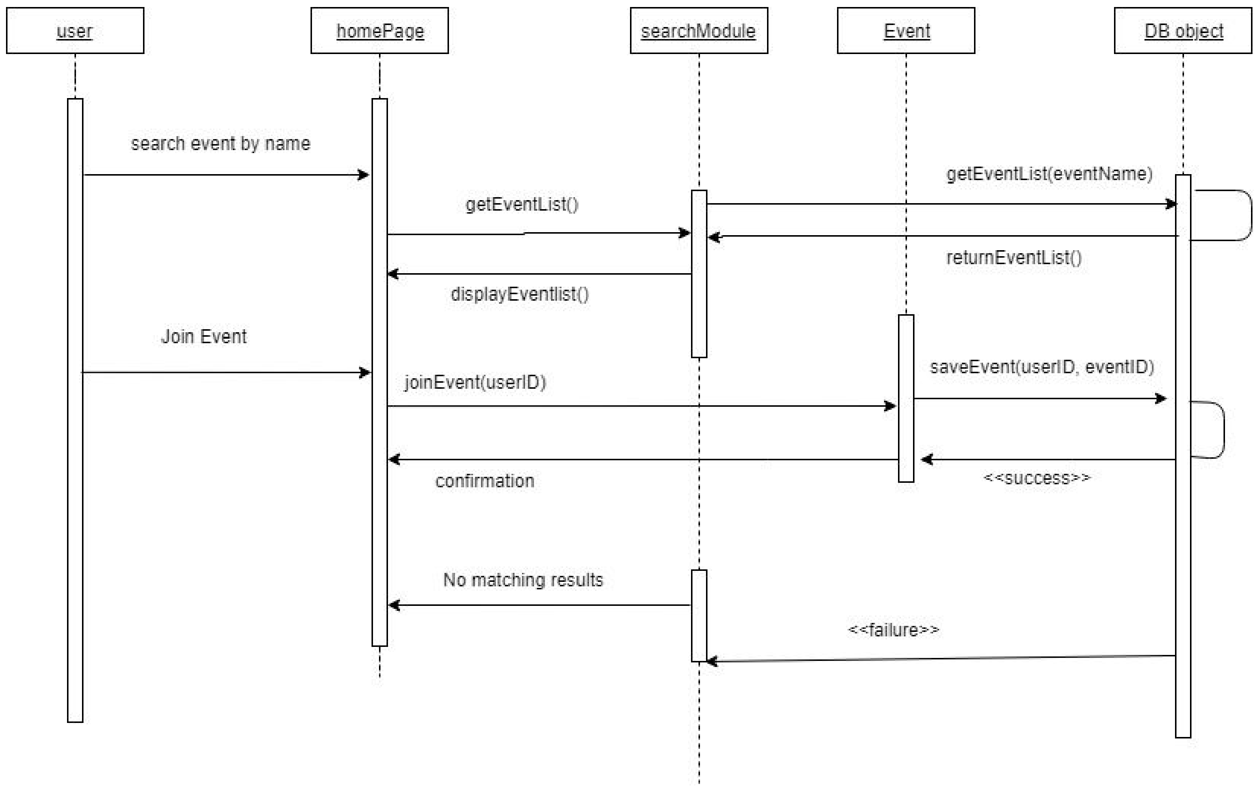


2.4 Sequence Diagrams

2.4.1 Registration



2.4.2 Search for Event


user
homePage
searchModule
Event
DB object
search event by name
getEventList(eventName)
getEventList()
returnEventList()
displayEventlist()
Join Event
saveEvent(userID, eventID)
joinEvent(userID)
confirmation
<<success>>
No matching results
<<failure>>